HAL open science

Génération d'ontologie à partir d'un modèle métier UML annoté

Cyril Faucher, Frédéric Bertrand, Jean-Yves Lafaye

To cite this version:

Cyril Faucher, Frédéric Bertrand, Jean-Yves Lafaye. Génération d'ontologie à partir d'un modèle métier UML annoté. Revue des Nouvelles Technologies de l'Information, 2008, 12, pp.65–84. inria-00460298

HAL Id: inria-00460298

https://inria.hal.science/inria-00460298v1

Submitted on 26 Feb 2010

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Génération d'ontologie à partir d'un modèle métier UML annoté

Cyril Faucher*, Frédéric Bertrand**
Jean-Yves Lafaye**

*IRISA/INRIA – Campus de Beaulieu
35042 Rennes, France
cfaucher@irisa.fr
http://www.irisa.fr
**Laboratoire L3i – Université de La Rochelle
17042 La Rochelle Cedex 01
frederic.bertrand@univ-lr.fr, jean-yves@lafaye
http://www-l3i.univ-lr.fr

Résumé. Dans cet article nous montrons de quelle manière les techniques d'ingénierie dirigée par les modèles peuvent être utilisées dans un processus de création d'une ontologie. À partir d'un modèle métier UML, déjà existant, et annoté avec un ensemble de stéréotypes, nous décrivons les différentes phases de la génération de l'ontologie. Le processus de création permet également, à partir d'objets métier, de peupler une base de connaissance structurée par l'ontologie générée. Ces travaux s'appuient sur la spécification ODM dont nous présentons les principes pour expliquer les concepts qui ont guidé notre démarche. Ce travail a été développé dans le domaine du journalisme pour produire une ontologie nécessaire à la navigation entre les différentes informations fournies par des dépêches.

DRAFT

1 Introduction

Dans de nombreux domaines d'activité utilisant des systèmes d'information, la connaissance liée à un métier est souvent représentée par des modèles UML (OMG_2, 2004) notamment par des diagrammes de classes modélisant les entités propres au domaine. Ces abstractions, appelées « modèles métier », capitalisent un ensemble de connaissances exprimées dans un langage largement répandu et normalisé. Ce formalisme présente l'avantage d'être maîtrisé par une importante communauté et constitue ainsi un moyen d'échange privilégié entre les parties qui conçoivent un logiciel. Notre démarche est de montrer que les connaissances présentes dans ces modèles peuvent, grâce à un processus d'annotation, permettre la production d'ontologies. La méthodologie que nous avons définie permet d'alléger et de guider le travail important que constitue la construction complète d'une ontologie en réutilisant des connaissances déjà acquises par l'expertise métier.

Dans le cadre du projet RIAM[1] « Relaxmultimédia » mené en collaboration avec deux agences de presse, l'Agence France-Presse et Relaxnews, nous avions pour objectifs de construire un environnement de production et de consultation de dépêches d'agence de presse, et plus particulièrement de capitaliser l'information apportée chaque jour par les dépêches d'agences dans une base de connaissance. Pour cela, il est apparu nécessaire de construire une ontologie relative au contenu des dépêches. Cette ontologie devant servir de référentiel à un outil de navigation développé dans le cadre du projet, il était important qu'elle puisse être construite à partir des connaissances métier déjà existantes et par conséquent du modèle métier déjà défini.

Une base de connaissance a été utilisée pour stocker des informations sur les concepts traités par les journalistes. Bien que ces informations soient accessibles via des systèmes de persistance classique (SGBD), ces derniers, privilégiant la structuration et la cohérence des données manquent de flexibilité et s'avèrent peu adaptés à la navigation sémantique. Les outils d'interrogation et de navigation du web sémantique, se connectant sur des bases de connaissance, ont l'avantage de tirer parti des relations entre concepts, reposant sur le paradigme du triplet d'information (objet, prédicat, valeur), tout en relâchant certaines contraintes propres aux langages de modélisation. Par exemple,, une base de connaissance autorisera, deux concepts issus d'une même classe à posséder des séquences différentes d'attributs ou permettra qu'un concept apparaisse dans plusieurs hiérarchies de types différentes. En revanche un objet UML est instance d'une seule classe, et les objets d'une classe ont nécessairement tous la même structure. La navigation s'appuyant essentiellement sur les relations entre les concepts, la génération de l'ontologie doit pouvoir extraire ces relations d'un modèle métier existant. Certains attributs sont utilisés pour fournir des labels et des métadonnées sur ces concepts, mais la variabilité offerte par une ontologie sur les attributs et les types ne sont pas des critères déterminants et exploités dans notre cas. Ainsi, l'extraction des informations du domaine à partir d'un modèle de classes UML apparaît pleinement satisfaisante. Nous verrons par la suite, que les associations entre classes fournies par UML suffisent à exprimer les propriétés ciblées dans l'ontologie résultante.

Pour parvenir à ce résultat, nous nous sommes fondés sur la proposition ODM (*Ontology Definition Metamodel*) récemment produite par l'OMG (IBM et al, 2006). Cette proposition spécifie des métamodèles pour différents langages de représentation des connaissances comme RDF (Klyne et Caroll, 2004), OWL (Patel-Schneider et al, 2004) ou les Topic Maps (ISO, 1999) en utilisant le formalisme UML. À l'aide de ces métamodèles et des méthodes de transformation associées (profils et règles de correspondances), il est possible de transformer des modèles métier exprimés avec UML en ontologies se conformant à ces différents métamodèles. Néanmoins, les modèles métier contiennent un certain nombre d'informations qui ne présentent pas d'intérêt pour l'ontologie[2]. En effet, certaines classes et associations n'apportent pas de valeur en termes de connaissances. Il convient donc, par un processus d'annotation, de sélectionner les éléments (classes, attributs et associations) intéressants du modèle. Les annotations sont exploitées lors de la construction de l'ontologie pour « guider » le processus de transformation de modèles. L'ontologie ainsi produite peut alors être régénérée, en cas de modification du modèle métier, grâce à l'automatisation du processus.

[1] http://www.riam.org

[2] Dans notre travail, nous entendons le terme « ontologie » comme un modèle représentant les concepts d'un domaine et les relations existant entre ces concepts.

Conjointement à l'utilisation d'un modèle métier pour la production de l'ontologie, nous avons exploité la présence d'objets métier (instances) pour peupler la base de connaissance structurée par l'ontologie générée. Ces objets métier sont créés de manière incrémentale et continue, guidée par l'arrivée de nouvelles informations. Cette approche permet ainsi de connaître et de contrôler l'origine des ressources introduites dans la base de connaissance.

La suite de cet article présente, dans la section 2, les principes essentiels de l'ingénierie dirigée par les modèles (IDM) en détaillant certains aspects de la transformation de modèles. Les concepts décrits dans cette section sont nécessaires pour appréhender les fondements et l'intérêt de la proposition ODM. Dans la section 3, nous présentons brièvement la proposition ODM dans ses aspects généraux puis nous détaillons les éléments que nous avons utilisés dans la définition de notre méthodologie. Notre méthodologie ainsi que le domaine d'application sont décrits dans la section 4. Dans la section 5 nous présentons et comparons notre approche avec des travaux connexes, puis nous concluons et présentons les perspectives de notre travail dans la section 6 en soulignant la généricité de la méthodologie définie et la possibilité de l'appliquer à d'autres domaines d'applications.

2 L'ingénierie dirigée par les modèles

2.1 Les principes

Parmi les notions essentielles présentes dans l'ingénierie dirigée par les modèles, il existe deux concepts fondamentaux : ceux de *modèle* et de *métamodèle*. On utilise un modèle pour abstraire la réalité, parfois complexe, dans le cadre d'un but opérationnel déterminé. Dans le vocabulaire de l'IDM, un système est *représenté* par un modèle. L'introduction du concept de métamodèle permet de définir des caractéristiques communes à un ensemble de modèles. Un métamodèle représente une spécification formelle d'une abstraction, généralement consensuelle et normative. Le concept de métamodèle permet également d'aider, pour un système donné, à la construction d'un modèle. Un modèle est lié à son métamodèle par une relation de *conformité*.

L'illustration de ces principes se retrouve dans l'architecture à quatre couches proposée par l'OMG dans son approche MDA (*Model Driven Architecture*) décrite dans (Miller et Mukerji, 2003). Cette architecture (cf. Fig. 1) est constituée, au niveau le plus bas, par la couche *M0* représentant le système réel. Un modèle représente ce système au niveau *M1*. Ce modèle se conforme à son métamodèle définit au niveau *M2* et le métamodèle se conforme au méta-métamodèle au niveau *M3*. Le méta-métamodèle se conforme à lui-même.

Un des intérêts majeurs de l'architecture MDA (Bézivin, 2005) est l'existence de l'auto-définition du méta-métamodèle au niveau *M3*. Grâce à l'existence de cette structure de référence commune, il est possible d'établir des relations entre modèles conformes à différents métamodèles. Un exemple de telles relations est fourni par la transformation de modèles. En effet, dans l'approche MDA, l'objectif est de générer des modèles spécifiques à une plate-forme (PSM, *Platform-Specific Model*) à partir de modèles indépendants de toute plate-forme (PIM, *Platform-Independent Model*). Dans ce but, il a été spécifié (OMG_5, 2005) le langage QVT (*Query, View, Transform*) qui permet l'expression de telles transformations.

Le choix, dans l'approche MDA, de la recommandation MOF (*Meta Object Facility*) (OMG_6, 2006) comme méta-métamodèle au niveau *M3*, permet l'utilisation d'un ensemble de métamodèles standardisés tels qu'UML, CWM (OMG_1, 2003), SPEM (OMG_3, 2005)...

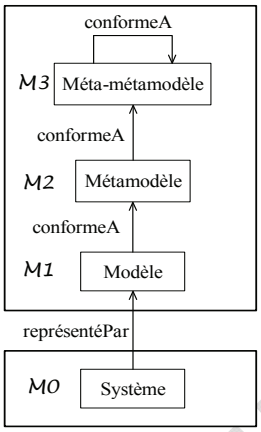

FIG. 1 – *L'organisation MDA en quatre couches*

Il permet également de définir des projections entre l'espace technique MDA et d'autres espaces techniques comme celui d'XML (exemple du standard XMI (OMG_4, 2005) utilisé pour l'échange de modèles) ou celui de Java (exemple de la spécification JMI (Dirckze, 2002) définissant une API Java pour la manipulation de modèles).

2.2 Les techniques et les outils

Un aspect important de l'ingénierie des modèles est représenté par la transformation de modèles. De manière générale, la transformation de modèles utilise un ensemble de programmes prenant en entrée des modèles (accompagnés de leur métamodèle) et produisant en sortie d'autres modèles, comme illustré par la Fig. 2.

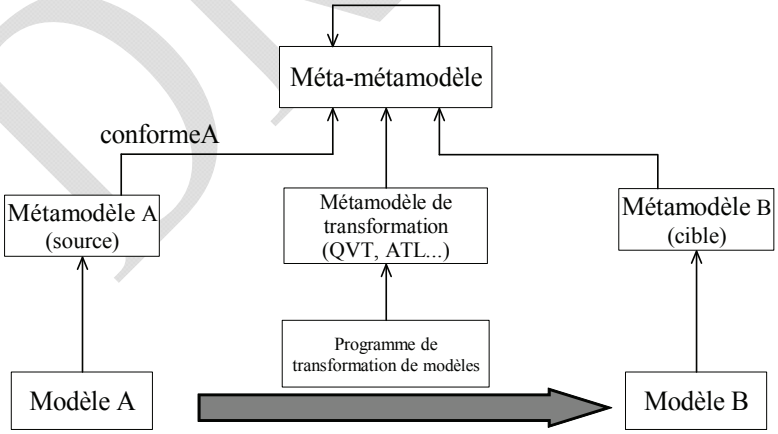

FIG. 2 – *La transformation de modèles*

DRAFT

Il existe de nombreux outils permettant de définir des transformations parmi lesquels on peut distinguer quatre catégories :

1. les outils de transformation génériques, catégorie dans laquelle nous trouvons, par exemple, les outils appartenant à l'espace technique XML comme XSLT (Clark, 1999) ou XML Query (Boag et al, 2006) ;
2. les langages de script intégrés à la majorité des ateliers de génie logiciel comme le langage J de l'atelier Objecteering[3] ;
3. les outils dédiés spécifiquement à la transformation de modèles et conçus pour être intégrables dans des environnements de développement normalisés. Comme exemples, on peut citer le langage ATL (*Atlas Transformation Language*) (Bézivin et al, 2003) et QVT ;
4. les outils de métamodélisation dans lesquels la transformation de modèles correspond à l'exécution d'un métaprogramme. Celui-ci manipule les modèles et métamodèles d'entrées et de sorties grâce à la réflexivité du langage. La transformation est implémentée en ajoutant aux parties structurelles des comportements grâce à un langage d'action. Le langage Kermeta, (Muller et al., 2005), est un exemple de langage appartenant à cette catégorie.

Bien que l'objectif de cet article ne soit pas une étude comparative des langages appartenant à ces différentes catégories, nous pouvons noter que les langages issus de la première catégorie possèdent l'inconvénient de ne pas pouvoir travailler au niveau de la sémantique des modèles manipulés. Ceux de la seconde catégorie sont essentiellement propriétaires (l'investissement est fortement dépendant de l'évolution de ces outils) et ont souvent été conçus comme des langages utilitaires et non pensés en termes de réels langages de transformation. Les deux dernières catégories permettent de disposer pleinement de langages efficaces pour réaliser des transformations, car leur syntaxe est orientée modèle et non objet.

3 La proposition ODM

La spécification ODM définit un ensemble de métamodèles d'ontologie, de profils associés et de mise en correspondance. L'objectif poursuivi est de permettre l'échange des connaissances définies selon des formalismes différents. Il existe deux possibilités spécifiées dans ODM pour le passage d'un modèle UML vers un modèle RDFS (Brickley et Guha, 2004) : soit en utilisant des profils UML, l'approche que nous avons suivie, soit par écriture de règles de transformations en utilisant le langage QVT. Dans ODM, le choix a été fait de construire un ensemble de métamodèles conformes au MOF pour définir la sémantique de plusieurs langages de représentation des connaissances et ainsi faciliter l'échange de modèles de connaissance.

Le choix d'UML pour décrire des ontologies pourrait paraître comme une alternative beaucoup plus simple. En observant rapidement les langages de description d'ontologies et UML on peut constater l'existence d'un certain nombre de notions communes : classes, relations, propriétés, héritage… Néanmoins, il existe plusieurs différences significatives entre ces concepts. La plus importante porte sur la notion de propriété – dans UML un attribut a une portée liée à la classe, à la différence des ontologies dans lesquelles une propriété représente un concept de premier niveau pouvant exister indépendamment d'une classe. D'autres

[3] Softeam, http://www.objecteering.com

différences portent sur un certain nombre de limitations d'UML, notamment en ce qui concerne la définition de propriétés sur les ensembles (disjonction, complément) qui empêche l'utilisation d'outils de déduction automatique sur des modèles UML.

ODM propose trois métamodèles (RDFS, OWL et Topic Maps) pour les formalismes les plus utilisés actuellement dans la communauté du Web sémantique. Les profils UML existant dans ODM fournissent une passerelle entre UML et les différents langages de représentation de connaissances. Un profil UML, dénommé également métamodèle virtuel, est un ensemble de stéréotypes et de valeurs étiquetées qui définissent des points d'extension du métamodèle d'UML. Les stéréotypes peuvent s'appliquer sur la plupart des éléments de modèles tels que les classes, attributs ou encore associations. Ces stéréotypes permettent de distinguer et de spécialiser les éléments de modèles en leur donnant une sémantique particulière. Les stéréotypes sont associés à des valeurs étiquetées qui se composent d'un nom de propriété et d'une valeur associée. L'application des stéréotypes sur un modèle métier est aussi appelée phase d'annotation. L'ingénierie des modèles rend possible l'interprétation des stéréotypes par l'intermédiaire de programmes informatiques. Ainsi l'implémentation de logiciels peut être dirigée par l'utilisation de stéréotypes et de notations communes à travers des outils de modélisation UML.

Dans la définition de notre méthodologie nous avons utilisé le profil UML pour RDF afin de définir nos différentes transformations. Celles-ci permettent, à partir du modèle métier exprimé en UML, de construire notre ontologie exprimée avec RDFS, en s'appuyant sur les principales correspondances décrites dans le tableau ci-contre (cf. TAB. 1).

UML	RDFS
Class	`rdfs:Class`
Generalization	`rdfs:subClassOf`
Association	`rdf:Property`
Attribute	`rdf:Property`
InstanceOf	`rdf:type`
Attribute type : String	`rdfs:Literal`
Attribute value	`rdf:value`

TAB. 1 - *Correspondances utilisées entre les métamodèles UML et RDF Schema*

4 La génération d'ontologie dirigée par les modèles

4.1 Le domaine métier de l'IPTC

Dans le domaine de la presse, l'IPTC[4] représente un consortium formé par les plus importantes agences de presse mondiales (AFP, Reuters, AP…). Son rôle est de développer des standards pour l'échange d'informations. Dans ce but, l'IPTC a défini une classification des concepts utilisés par les journalistes sous forme d'un modèle de classes UML (cf. FIG. 3) représentant le modèle d'entrée de notre processus de génération.

Lors de la rédaction de dépêches, le journaliste fait référence à des concepts, que l'on appelle ici « entités nommées » (`NamedEntity`) et qui peuvent correspondre à des lieux (`Loca-`

[4] International Press Telecommunication Council.

`tionConcept`), des personnes (`PersonConcept`), des organisations (`OrganisationConcept`), des institutions (`InstitutionConcept`), des événements (`EventConcept`) ou des objets (`WorkConcept`). Afin d'accroître la connaissance sur ces entités, le système d'information existant offre au journaliste la possibilité de leur ajouter des informations. Les attributs concernant chaque type d'entités font appel à une autre partie du modèle métier que nous ne traitons pas dans cet article. En revanche, les relations que le journaliste peut indiquer entre des entités se révèlent très riches et ces informations sont stockées par des associations entre sous-classes de la classe `NamedEntity`.

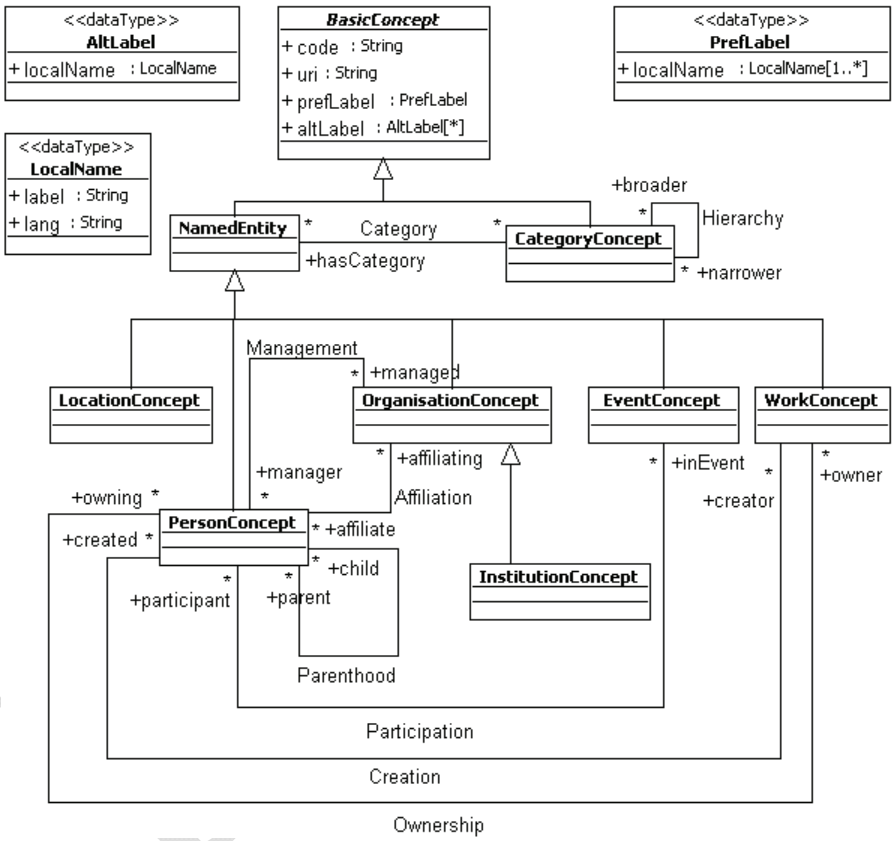

FIG. 3 – *Classification des concepts (extrait du modèle métier simplifié)*

Entre chaque type de concept, des associations, telles que `Participation` ou encore `Creation`, permettent de lier les entités entre elles. Une association réflexive sur la classe `NamedEntity` aurait pu permettre de lier les concepts, mais le souhait d'avoir une sémantique forte à travers les associations, nous oblige à spécifier les associations directement entre les classes concernées. Il est à noter que chaque concept peut possèder des `CategoryConcept` afin de catégoriser les entités. Les catégories, issues de l'IPTC, représentent une classification orthogonale à la hiérarchie de concepts issue de `NamedEntity`. Nous trouvons ici des centres d'intérêts comme les arts, la culture, les sports et les loisirs. Il existe des hiérarchies

de catégories organisées avec les rôles `broader` et `narrower`. Les catégories sont un des critères prépondérants pour faciliter la navigation entre les informations issues des dépêches.

4.2 Processus de génération de l'ontologie et de la base de connaissance

Le processus de génération (cf. FIG. 4) que nous avons développé permet la génération de l'ontologie ainsi que le peuplement de la base de connaissances. Ce processus intègre trois phases, une première d'annotation, les deux dernières représentant des transformations de modèles :

1. une première étape d'annotation est réalisée par un expert du métier, il choisit les éléments du modèle métier à introduire dans l'ontologie par l'intermédiaire de stéréotypes.
2. la seconde étape consiste à produire un modèle UML « pivot » à partir du modèle métier annoté.
3. la troisième étape produit l'ontologie dans un formalisme particulier, en l'occurrence RDFS.

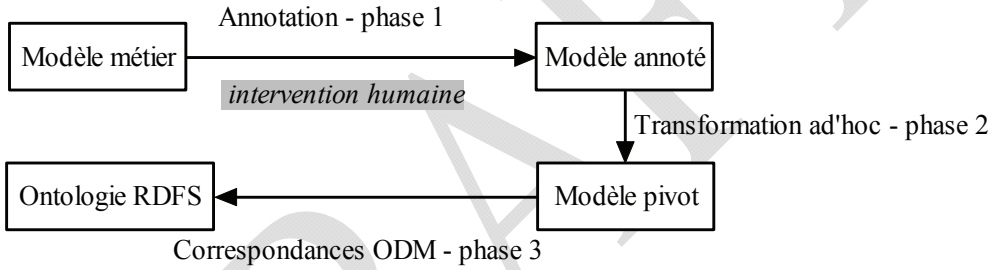

FIG. 4 – *Processus de transformations de modèles utilisé.*

Une génération directe du modèle métier vers une ontologie n'est pas satisfaisante, car cela introduit des éléments non pertinents pour les journalistes quant à la navigation dans la base de connaissance (e.g. les formats d'images, les droits d'accès aux ressources). Ainsi, un moyen de sélectionner les éléments métier pertinents est de mettre en place un processus de capture des informations, dit processus d'annotation. Dans notre cas d'application, l'expert choisit d'utiliser les annotations en fonction des besoins de navigation des journalistes. Ces besoins peuvent être capturés sous forme de cas d'utilisation validés par les utilisateurs.

Ce processus d'annotation utilise les mécanismes d'extension d'UML, i.e. : stéréotypes et valeurs étiquetées. L'annotation réalisée par l'expert métier indique le choix des concepts et relations qu'il souhaite retrouver dans l'ontologie et dans la base de connaissance. Le modèle annoté est ainsi une extension du modèle métier. Cette tâche est la plus importante et nécessite toute la connaissance de l'expert métier. Nous avons conçu les annotations de manière à masquer le langage cible de représentation des connaissances en créant des stéréotypes conformes au métier, ce qui diffère des profils UML d'ODM qui sont exprimés avec les termes réservés aux ontologies. En effet, les éléments d'annotation se veulent résolument sémantiquement proches du domaine traité, par exemple les notions de `Concept` et de `ConceptLink` décrits par la suite. L'alignement sur les métamodèles d'ODM et la terminologie du Web Sémantique intervient lors de la transformation produisant le modèle « pivot ».

Ce modèle est indépendant de tout langage de représentation des connaissances. La FIG. 5 montre le schéma général des différentes transformations mises en œuvre.

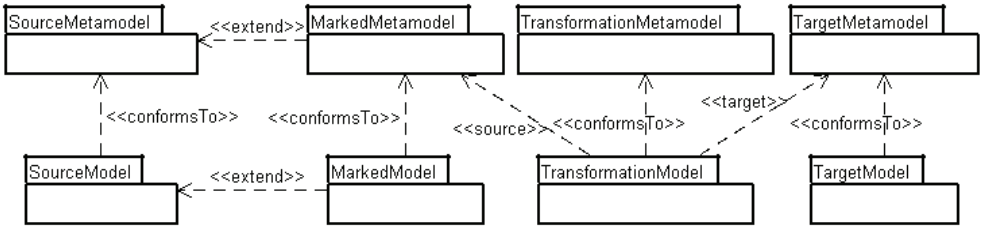

FIG. 5 – *Transformation de modèles dirigée par les métamodèles*

Le modèle métier annoté fait l'objet d'une première transformation vers un modèle UML « pivot » prêt à la transformation vers un langage de représentation spécifique, dans notre cas RDFS. Le modèle « pivot » contient les informations nécessaires à la création de l'ontologie, il est conforme au métamodèle UML. Le modèle « pivot » résultant aurait également pu être créé directement, sans passer par la phase d'annotation puis de transformation. Dans notre cas ce raccourci n'est pas envisageable. D'une part l'objectif était d'utiliser les termes du domaine, ceci impliquant la création d'une couche d'abstraction supplémentaire et, par conséquent, l'ajout d'une étape. D'autre part, nous traitons à la fois du modèle et de ses instances. Ces instances ne sont évidemment pas connues lors de la phase de modélisation, il était donc plus judicieux de définir un algorithme de transformation utilisable aux deux niveaux : ontologie et base de connaissance.

La seconde transformation produit l'ontologie à partir de ce modèle « pivot » en utilisant les correspondances fondées sur celles d'ODM entre les éléments UML et les éléments d'une structure d'ontologie comme RDFS (cf. TAB. 1). Ainsi l'annotation du modèle métier par l'expert, dirige la constitution du modèle pivot et finalement la construction de l'ontologie.

Nous avons ici privilégié la réutilisabilité et l'utilisation de standards comme ODM ; cette même structure UML peut nous servir à générer plusieurs ontologies utilisant des formalismes différents. Par exemple, comme nous possédons un modèle UML représentant l'ontologie et qu'ODM fournit les transformations d'UML vers RDFS et OWL, nous pouvons générer une ontologie exprimée avec RDFS et une autre avec OWL.

De manière similaire au processus de génération de l'ontologie, le processus de peuplement de la base de connaissance fait appel à deux phases de génération. Ces transformations de modèles sont « liées » par la relation d'instance entre les classes et les objets. Plus précisément (cf. FIG. 6), il existe pour l'ontologie une transformation des classes métier vers une représentation des `rdfs:Class` dans le modèle pivot. Pour les instances, il existe le même type de correspondance qui transforme une instance métier en une instance du modèle « pivot ». Il est à noter que les annotations, définies pour la génération de l'ontologie, sont également celles utilisées pour le peuplement de la base de connaissance.

Toute modification de l'annotation sur le modèle métier est prise en compte lors de la régénération de la base de connaissance. En effet, l'ensemble des instances du modèle métier sont conservées et peuvent être interrogées à tout moment. Ainsi, notre démarche offre une traçabilité entre un objet UML et l'élément RDF correspondant dans la base de connaissance.

DRAFT

Il faut cependant noter que cette approche est unidirectionnelle, i.e. : des modifications qui seraient effectuées dans la base de connaissance ne seront pas reportées sur le modèle métier.

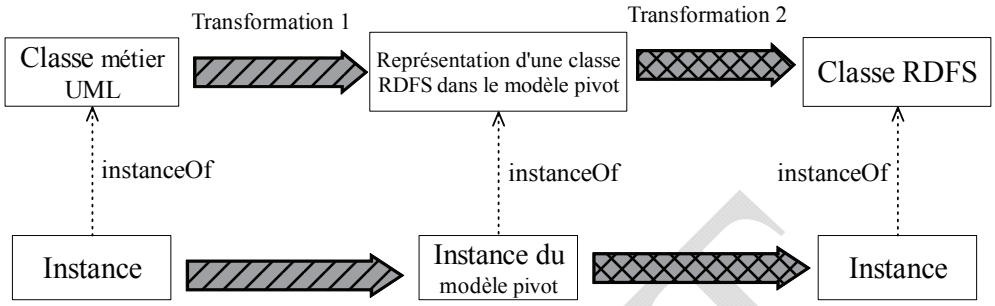

FIG. 6 – *Les deux transformations du processus de génération*

4.3 Définition du profil UML utilisé pour l'annotation

Afin de proposer un modèle standardisé de thésaurus, le consortium W3 a récemment proposé un vocabulaire RDF permettant de construire un système simple pour l'organisation des connaissances : SKOS, *Simple Knowledge Organisation System*, (SKOS, 2004). Ce modèle est constitué de concepts et de relations entre ces concepts.

Ces relations peuvent être :

- hiérarchiques (`skos:narrower` et `skos:broader`) permettant la navigation via la propriété de subsomption ;
- associatives (`skos:related`) en permettant d'associer deux concepts.

Il existe également des propriétés permettant de préciser le nom principal d'un concept (`skos:prefLabel`) ou son nom alternatif (`skos:altLabel`).

Le métamodèle de thésaurus utilisé s'appuie sur celui de SKOS en reprenant notamment la notion de concept. Tout concept est sous-classe (`rdfs:subClassOf`) de la classe RDF `Concept`.

À cet effet, nous souhaitions extraire les classes et leurs propriétés : attributs et rôles. Toutes les classes du domaine ne sont pas à intégrer dans l'ontologie, uniquement certaines sont intéressantes. De manière similaire, toutes les associations entre classes intéressantes ne sont pas nécessairement à intégrer dans l'ontologie. Nous devions également gérer les aspects multilingues pour les noms de propriétés d'association et l'utilisation de l'inférence pour certaines propriétés notamment `skos:broader` à laquelle nous avons donné la propriété de transitivité.

Les stéréotypes inclus dans le profil UML sont issus d'une réflexion avec les experts du domaine.qui en ont assuré la validation.. Notre travail a été de formaliser ces concepts pour les rendre interprétables par un programme informatique.

Notre contribution se distingue de celle de SKOS par la modélisation des relations entre les concepts pour lesquelles nous avons défini la notion de « `conceptLink` ». À noter que les « `conceptLink` » ne sont pas directement liés aux concepts. En effet, comme dans le métamodèle d'UML, nous utilisons la notion de propriété pour modéliser les rôles d'un concept, ce sont des « `conceptProperty` ». Les `conceptLink` sont ainsi liés aux concepts par

DRAFT

l'intermédiaire des `conceptProperty`. Un concept peut avoir plusieurs propriétés elles-mêmes liées à un `conceptLink`.

Les stéréotypes liés à la génération de l'ontologie sont présentés ci-dessous (cf. **FIG. 7** et cf. TAB 2).

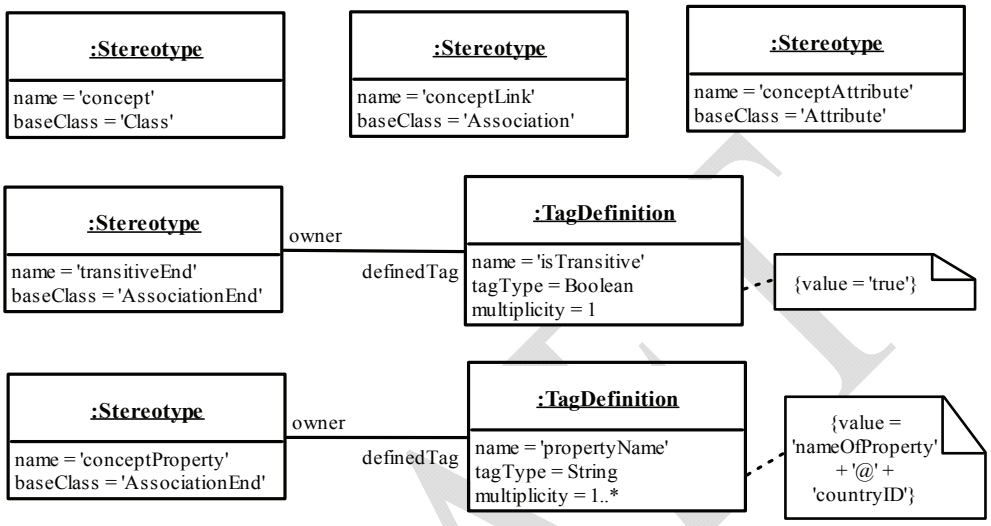

FIG. 7 – *Définitions des différents stéréotypes utilisés pour l'annotation*

Nom	**S'applique à**	**Tag** (valeur étiquetée)	**Description**
`concept`	`Class`		Classe représentant un concept que l'on souhaite intégrer dans l'ontologie
`conceptAt-tribute`	`Property`		Propriété représentant un attribut d'une classe stéréotypée `concept`
`conceptLink`	`Association`		Association, entre deux classes `concept`, sémantiquement intéressante pour intégrer l'ontologie
`conceptPro-perty`	`AssociationEnd`	`propertyName` de type `String` avec une cardinalité multiple (1..*). Les valeurs étiquetées correspondantes vont contenir les noms des rôles dans une langue donnée sous la forme : `nameOfTheProperty@`**`en`**	Extrémité d'association liée à une classe `concept` et à une association `conceptLink`, la valeur du nom de l'extrémité d'association contient un code. `conceptProperty` possède des valeurs étiquetées.

TAB. 2 – *Stéréotypes liés à la génération de l'ontologie*

DRAFT

Le stéréotype lié à la génération de la base de connaissance est présenté sur le TAB. 3.

Nom	S'applique à	Description
`transitiveEnd`	`AssociationEnd`	Extrémité d'association `conceptProperty` possèdant la particularité d'être transitive : lors de la génération de la base de connaissance une inférence de transitivité va être appliquée, ex. : relation « *broader* ».

TAB. 3 – *Stéréotype lié à la génération de la base de connaissance*

La FIG. 8 présente deux cas d'utilisation pour la mise en œuvre de l'annotation avec les stéréotypes définis précédemment. Le premier cas d'utilisation (partie gauche) spécifie le fait que le journaliste souhaite naviguer d'un `WorkConcept` vers une `PersonConcept` en utilisant le rôle `creator`. Ceci se traduit par l'application aux classes `WorkConcept` et `PersonConcept` du stéréotype « `concept` ». L'association `Creation` doit être stéréotypée « `conceptLink` » afin de prendre en compte le rôle `creator`. L'utilisateur souhaitant naviguer suivant le rôle `creator`, ainsi il est stéréotypé « `conceptProperty` ». Dans le second cas d'utilisation (partie droite), les deux rôles de l'association `Creation` sont stéréotypés « `conceptProperty` ». Dans ce cas il est indiqué que l'on souhaite naviguer de manière bi-directionnelle entre des `WorkConcept` et des `PersonConcept`. Ces deux exemples d'annotation montrent également la possibilité de filtrer les attributs utilisables pour la navigation et la recherche d'objets dans la base de connaissance. Par exemple, l'attribut `altLabel` de la classe `BasicConcept` n'est pas stéréotypé dans le premier exemple, mais le second. Ainsi les labels alternatifs seront disponibles pour l'affichage de métadonnées et la recherche de concepts dans la base. Les classes stéréotypées par « `concept` » seront introduites dans l'ontologie comme de type `rdfs:Class` et comme sous-classes de `Concept`. Les classes `PersonConcept` et `WorkConcept` héritent de `BasicConcept`, et, de ce fait, leurs `rdfs:Class` correspondantes seront des sous-classes de `BasicConcept`. Les attributs stéréotypés « `conceptAttribute` » seront accessible soit sous forme de littéraux comme pour `code` et `uri`, soit sous forme de propriétés pointant vers `PrefLabel` (*Preferred Label*) ou `AltLabel` (*Alternative Label*) définies en 4.1.

DRAFT

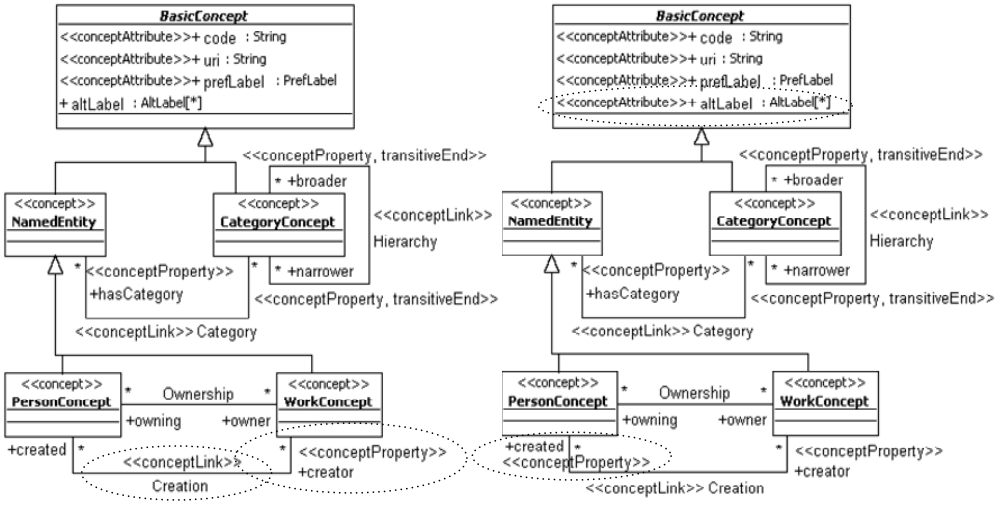

FIG. 8 – *Extrait du modèle métier annoté*

Les classes `PersonConcept` et `WorkConcept` sont reliées par deux associations : `Creation` et `Ownership`, seule `Creation` est stéréotypée « `conceptLink` ». Par conséquent, seule l'association `Creation` va participer à la génération de l'ontologie. Les rôles `creator` et `created` vont devenir deux `rdf:property`, respectivement nommées `WorkConcept-creator-PersonConcept` et `PersonConcept-created-WorkConcept`. L'URI d'une `rdf:property` est la concaténation de `domain` + `label` + `range`, ceci pour résoudre des problèmes d'unicité, car notre modèle utilise plusieurs fois un même libellé de rôle. Des libellés (`Property.name`) sont ajoutés pour documenter ces nouvelles `rdf:property` afin de disposer d'un libellé plus expressif qu'une concaténation de termes et de traductions dans différentes langues. Ces libellés sont stockés dans les valeurs étiquetées (cf. FIG. 8) associées au stéréotype « `conceptProperty` » : ce sont des valeurs étiquetées nommées `propertyName` dont la valeur est la concaténation `libellé` + `@` + `identifiantLangue` par exemple : `creator@en` ou `créateur@fr`.

La relation de `Hierarchy` entre deux instances de `CategoryConcept` possède deux rôles : l'un pour exprimer la relation de `broader` et l'autre de `narrower`, ces deux derniers sont stéréotypés à la fois comme « `conceptProperty` » et comme « `transitiveEnd` ». Ainsi `broader` et `narrower` deviendront des `rdf:property`. L'information indiquant la propriété de transitivité est ajoutée dans le fichier de configuration du moteur d'inférence de Sesame ; notre base de connaissance.

4.4 Les différents modèles générés

Nous montrons, sur la FIG. 9, un extrait du modèle pivot généré sous forme d'un diagramme de classe UML qui montre le résultat de la première transformation : modèle annoté vers modèle « pivot ». Les éléments `NamedEntity`, `PersonConcept` et `WorkConcept` ont été conservés du fait de leur statut de « concept ». `BasicConcept` est également présent avec ses attributs qui étaient stéréotypés « `conceptAttribute` ». L'association « `creation` » entre les classes `PersonConcept` et `WorkConcept` a produit deux relations entre concepts :

`PersonConcept-created-WorkConcept` et `WorkConcept-creator-PersonConcept`. Il est à noter que l'aspect multilingue, avec les différentes traductions des libellés, n'est pas affiché sur cette figure, mais il est bien présent au sein du modèle.

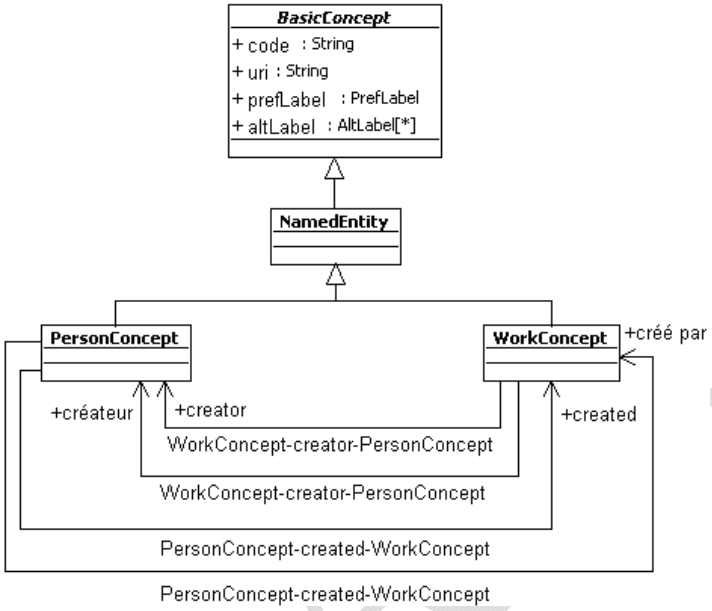

FIG. 9 – *Extrait du modèle « pivot » exprimé avec UML*

La FIG. 10 montre le résultat de la génération de l'ontologie RDFS. On peut remarquer que l'association `Ownership` n'apparaît pas car seules les associations stéréotypées « `conceptLink` » et les rôles stéréotypés « `conceptProperty` » sont utilisés pour générer l'ontologie. Les `rdfs:Class PersonConcept` et `WorkConcept` sont des sous-classes de `Concept`. Il est à noter que la `rdfs:Class NamedEntity` n'est pas représentée pour des raisons de place.

La FIG. 11 montre le résultat de la génération d'une base de connaissance grâce à l'outil développé. On peut observer les deux instances insérées dans la base de connaissance que sont `Léonard de Vinci` et `La Joconde`. La ressource `Léonard de Vinci` est insérée comme un `PersonConcept` et `La Joconde` comme un `WorkConcept`. (`rdf:type` associé à `PersonConcept` et à `WorkConcept`). On peut également remarquer l'organisation des libellés préférés (`prefLabel`) et alternatifs (`altLabel`) sur `Léonard de Vinci` qui se nomme également `Leonardo di ser Piero da Vinci`. Les éléments `ns1:9` et `ns1:11` représentent les URI qui référencent respectivement `Léonard de Vinci` et `La Joconde`. Entre ces deux références, les deux propriétés `PersonConcept-created-WorkConcept` et `WorkConcept-creator-PersonConcept`, issues des propriétés entre les classes RDFS `PersonConcept` et `WorkConcept`, permettent une navigation bidirectionnelle entre ces ressources.

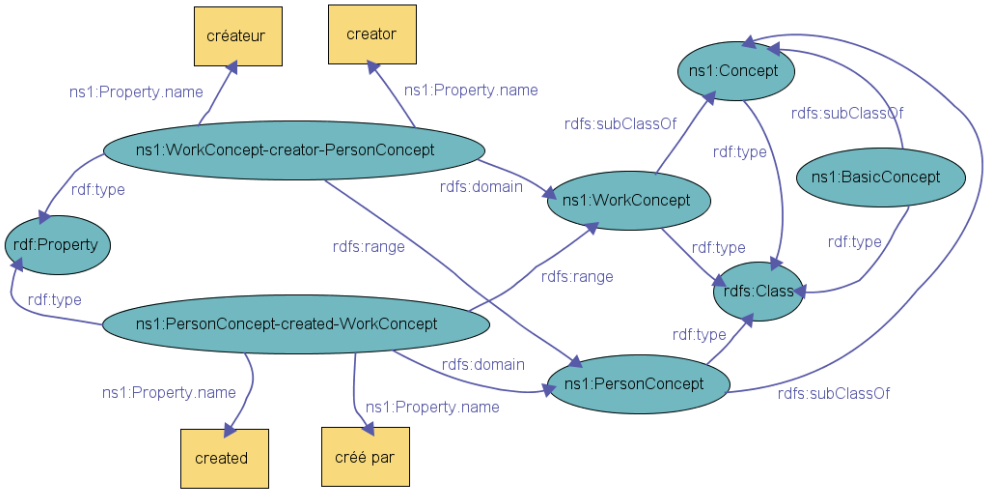

FIG. 10 – *Extrait de l'ontologie générée*

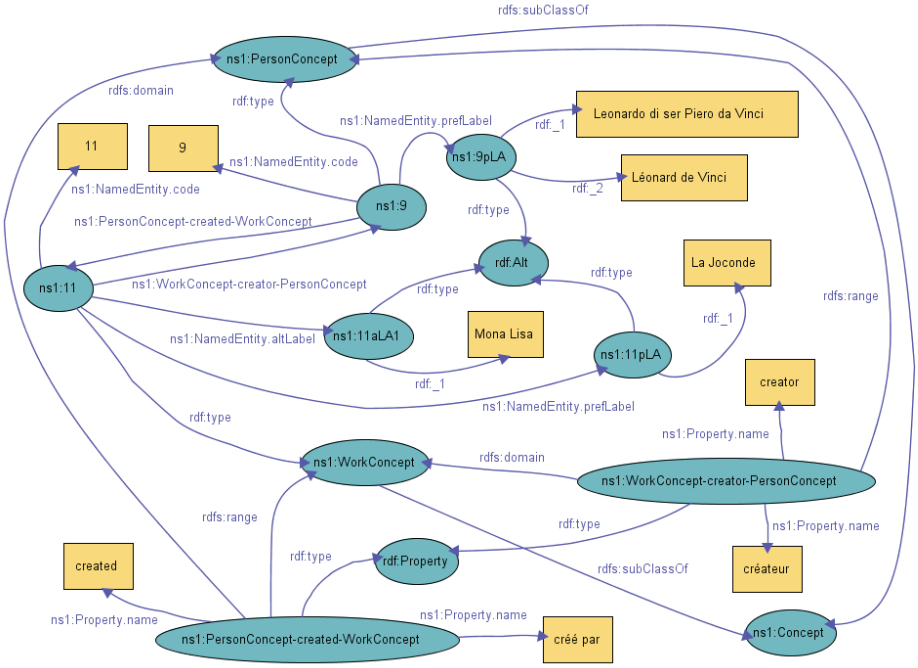

FIG. 11 – *Extrait de la base de connaissance générée*

Concernant les objets métier et le peuplement de la base de connaissance, le processus se déroule de la manière suivante. Les dépêches sont produites selon un format XML. Ce flux XML est traité par un outil de catégorisation qui a un double rôle :

1. à partir d'une base de connaissance il reconnait et extrait du texte des dépêches les concepts déjà référencés ;
2. il propose certains mots (qui ont un indice de pertinence élevé) à l'administrateur de la base de connaissance pour être validés et devenir des concepts.

Parallèlement à ce processus, des objets métier sont créés et maintenus (mise à jour d'information), leurs informations étant stockées dans une base de données. Nous avons choisi d'utiliser ces objets pour instancier le modèle pivot. La persistance des objets métier originaux est primordiale car elle permet de régénérer la base de connaissance sans se soucier de l'intégrité des données, dont la garantie est déléguée au SGBD plus apte à remplir cette tâche que la base de connaissance. Il est intéressant de générer de nouveau la base de connaissance lorsque l'annotation du modèle métier change, ou si les règles de transformations sont modifiées, ou bien encore si l'on souhaite viser un autre entrepôt RDFS. Ainsi, les objets sont extraits d'une des deux sources : le flux des dépêches ou la base de données, puis ils sont transformés en instances du modèle pivot avec une application Java manipulant les modèles. Le modèle pivot étant instancié, alors une seconde transformation lance la génération des instances de l'ontologie (instances du schéma RDFS), i.e. : génération des triplets RDF dans l'entrepôt RDF Sesame. La propriété de transitivité exprimée sur la relation `broader` et, plus généralement, nos propres inférences ont été implémentées sous forme de règles dans la base RDF Sesame.

4.5 Mise en œuvre du processus de génération

Afin de manipuler les modèles et métamodèles nous avons utilisé l'API Java JMI (Dirckze, 2002) qui donne accès aux éléments du MOF. Pour stocker ces modèles et métamodèles nous avons utilisé l'outil MDR (Matula, 2003). Les entités UML : modèles et instances ont été manipulés grâce à l'API Java UML 1.4 générée à partir de JMI et nous avons utilisé l'API Java Sesame pour stocker le schéma RDF et les données RDF.

L'utilisation d'un langage dédié à la transformation de modèle, tel que Kermeta ou ATL, a été envisagée mais, à la date du début du projet, ces langages ne présentaient pas toute la maturité nécessaire ni les performances requises pour traiter des modèles de taille importante. De plus, les modèles métier initiaux étant disponibles sous l'atelier UML Poseidon[5], seul l'outil MDR s'est révélé compatible avec le format XMI d'exportation de Poseidon.

Pratiquement, le processus de génération de l'ontologie comporte cinq phases :

1. Le concepteur annote le modèle métier ;
2. Le modèle annoté est chargé à l'aide d'un analyseur de format XMI au niveau UML de la base de modèles MDR ;
3. Un premier outil de transformation que nous avons développé parcourt le modèle annoté pour générer le modèle pivot en exploitant les éléments stéréotypés et les valeurs étiquetées ;
4. Un second outil de transformation permet de générer l'ontologie, plus précisément le schéma RDFS ceci à partir du modèle pivot et des règles spécifiées dans ODM ;
5. La dernière étape est le chargement du schéma RDFS dans l'entrepôt RDF Sesame.

[5] http://www.gentleware.com

5 Travaux connexes

Il existe de nombreux travaux combinant UML et des langages de représentation d'ontologies. Parmi les travaux portant sur la génération d'ontologie dans une démarche d'ingénierie dirigée par les modèles :

- ceux de Gasevic (Gasevic1, 2006) et (Gasevic2, 2006). Ces travaux conduisent à la génération d'une ontologie à partir d'un modèle UML annoté par les stéréotypes composant le profil UML d'OWL et fournis par ODM. Les transformations sont réalisées par des feuilles de styles XSLT opérant sur des modèles au format XMI.. Dans ce cas précis, le modèle UML est une représentation sous forme de modèle de classes de l'ontologie visée. Le modèle reflète l'ontologie souhaitée. Notre travail se distingue de celui de Gasevic par l'annotation du modèle du domaine qui n'est pas « l'image UML » de notre future ontologie. En fait nous décalons la phase d'annotation en amont. Le résultat de l'annotation sera un modèle pivot, représentation UML de l'ontologie. Ce résultat intermédiaire est masqué de l'expert métier. À partir du modèle pivot nous pouvons, de la même manière que Gasevic, générer l'ontologie en utilisant les recommandations d'ODM. D'autre part, nous n'avons pas retenu le format XMI pour une transformation directe vers RDFS car la prise en compte des instances ne nous permet pas d'effectuer un simple « filtrage » pour produire le modèle RDFS. De plus différentes versions d'XMI existent et sont incompatibles entre-elles rendant l'utilisation de logiciels de modélisation.
- Dans (Brockmans, 2006) une démarche utilisant des techniques similaires (définition de profils UML et utilisation d'ODM) aux nôtres est présentée. Cependant l'objectif visé correspond à la visualisation d'ontologies OWL en utilisant la notation graphique d'UML, ce qui est assez éloigné de notre problématique.
- Nous pouvons également noter le projet Eclipse EODM[6] pour EMF *Ontology Definition Metamodel* qui se veut une implémentation du standard ODM en utilisant les technologies EMF (*Eclipse Modeling Framework*). EODM se propose d'offrir des éditeurs textuels et arborescents d'ontologies exprimées en RDFS et OWL. De plus, EDOM intègre une couche d'inférence et implémente les transformations UML vers RDFS et UML vers OWL par l'intermédiaire des APIs Java générées grâce à EMF.
- L'adaptation d'outils existants comme le module d'extension (*plugin*) UMLBackend[7] de l'éditeur d'ontologies Protégé[8]. Ce module permet l'importation et l'exportation de modèles UML 1.4 au format XMI mais possède de nombreuses limitations : la transformation n'est pas configurable, plusieurs éléments importants d'UML comme les stéréotypes et les valeurs étiquetées ne sont pas prises en compte.

À côté de ces approches nous pouvons souligner l'existence de moteurs de catégorisation comme TEMIS (Temis) ou encore des outils semi-automatiques comme CORESE (Corby et Faron, 2002) permettant d'annoter des documents à l'aide d'ontologies. Dans CORESE il est possible d'annoter directement des termes comme, par exemple Paris, qui sera annoté « Lieu » (`LocationConcept`). Pour notre part, nous annotons la classe métier `Location`

[6] http://www.eclipse.org/modeling
[7] http://protege.cim3.net/cgi-bin/wiki.pl?UMLBackend
[8] http://protege.stanford.edu

DRAFT

comme « concept » qui deviendra une classe en RDFS. Ainsi, toutes les instances (au sens UML) de la classe `Location` seront introduites dans la base de connaissance comme de type (au sens ontologie) `LocationConcept`. Comme il a été dit précédemment, notre démarche se veut très en amont, dès la phase de modélisation, lors de la capture des exigences et des connaissances métier.

6 Conclusion et perspectives

La méthodologie de construction d'ontologies que nous proposons est fondée sur l'annotation de modèles métier exprimés avec des diagrammes de classes UML. En appliquant des techniques développées dans le domaine de l'ingénierie des modèles, nous offrons la possibilité d'utiliser une partie des connaissances acquises sur un domaine métier pour construire une ontologie.

Néanmoins le modèle UML comprend des différences sémantiques relativement importantes par rapport aux différents modèles d'ontologies. Nous avons donc défini un profil UML constitué de stéréotypes et de valeurs étiquetées. Chaque élément du profil est associé à des règles de transformation visant à produire l'ontologie dans un format le plus indépendant possible des différents formalismes de représentation de connaissance. Ce format, nommé modèle « pivot », s'appuie sur les principes du modèle SKOS c'est-à-dire permettant une navigation hiérarchique et associative entre concepts. Le passage du modèle « pivot » vers l'ontologie est fondé sur les correspondances entre les modèles de classe UML et les ontologies proposées dans la spécification ODM. L'ontologie générée sert de support à la navigation dans la base de connaissance.

Notre contribution porte également sur la prise en compte des instances du modèle métier afin de peupler la base de connaissance. La transformation de modèles que nous avons définie s'applique ainsi à deux niveaux : au niveau modèle (*M1*), elle permet la génération de l'ontologie, au niveau des données (*M0*), elle permet d'utiliser les données du modèle métier pour construire la base de connaissance.

Notre approche est destinée à des systèmes d'information qui possèdent une vue objet de la connaissance métier et pour lesquels on souhaite extraire une ontologie et une base de connaissance. Notre méthode ne se substitue pas à des moteurs de catégorisation automatique de concepts, mais elle intervient juste après cette classification d'instances. Ce que nous avons décrit dans cet article peut naturellement s'appliquer à d'autres domaines d'activités que celui de la presse, les mémoires d'entreprises seraient sans doute un cas d'application intéressant. En effet, des travaux en cours (Khelif, 2006) exploitent ces mémoires afin de constituer des ontologies sur la connaissance métier des entreprises. Le contenu est généralement extrait de systèmes d'information existants et modélisés avec UML.

Nous travaillons actuellement sur la mise en œuvre d'une transformation[9] UML vers OWL à l'aide d'ATL pour mettre en œuvre les différentes transformations de modèles et inférences.

[9] http://www.eclipse.org/m2m/atl/usecases/ODMImplementation/

Références

Bézivin, J. (2005). *On The Unification Power of Models*. Software and System Modeling 4(2):171–188.

Bézivin, J., G. Dupé, F. Jouault, G. Pitette., and J. Rougui (2003). *First Experiments with the ATL Model Transformation Language: Transforming XSLT into XQuery*. OOPSLA 2003 Workshop. Anaheim, USA.

Boag, S., D. Chamberlin, M. Fernandez, D. Florescu, J. Robie, and J. Simeon (2006). *XQuery 1.0: An XML Query Language*. Recommandation candidate du W3C.

Brickley, D., and R. Guha (2004). *RDF Vocabulary Description Language 1.0: RDF Schema*. Recommandation W3C.

Brockmans, S., R. M. Colomb, E. F. Kendall, E. Wallace, C. Welty, G. T. Xie and P. Haase (2006). *A Model Driven Approach for Building OWL DL and OWL Full Ontologies*. 5th International Semantic Web Conference, vol. 4273 of LNCS, pp. 187-200, Athen, USA.

Broekstra J. (2005). *Storage, Querying and Inferencing for Semantic Web Languages*. PhD thesis, Vrije Universiteit. ISBN 90-9019-236-0

Corby, O. and C. Faron, (2002). *Corese : A Corporate Semantic Web Engine*, Proceedings of the International Workshop on Real World RDF and Semantic Web Applications, 11th International World Wide Web Conference, Hawai, USA.

Clark, J. (1999). *XSL Transformations (XSLT)*. Version 1.0. Recommandation W3C.

Dirckze, R. (2002). *Java Metadata Interface (JMI) Specification*. Version 1.0, JSR 040.

Gasevic$_1$, D., D. Djuric, and V. Devedzic (2006). MDA-based Automatic OWL Ontology Development. *International Journal of Software Tools for Technology Transfer* (à paraître).

Gasevic$_2$, D., D. Djuric, and V. Devedzic (2006). *Model Driven Architecture and Ontology Development*. Springer-Verlag.

IBM, and Sandpiper Software, Inc (2006). *Ontology Definition Metamodel.* Sixth Revised Submission to OMG/RFP ad/2003-03-40.

International Organization for Standardization (2000). ISO/IEC 13250, Information Technology — SGML Applications — Topic Maps.

Khelif, K., R. Dieng-Kuntz et P. Barbry (2006). "*Web sémantique pour la mémoire d'expériences d'une communauté scientifique : le projet MEAT*", Actes des 6ième journées d'Extraction et de Gestion de Connaissances (EGC'06), 175-186, Lille, France.

Klyne, G., J. Caroll (2004). *Resource Description Framework (RDF): Concepts and Abstract Syntax*. Recommandation W3C.

Matula, M. (2003). *NetBeans Metadata Repository*. http://mdr.netbeans.org/MDR-whitepaper.pdf

Miller, J., and J. Mukerji (2003). *MDA Guide*. Version 1.0.1.

DRAFT

Muller, P.A., F. Fleurey, and J.M. Jézéquel (2005). *Weaving executabillty into object-oriented meta-languages*. In Kent S. and Briand L. (eds), *Proceedings of MODELS/UML'2005*, volume 3713 of *LNCS*, pages 264-278. Montego Bay, Jamaica: Springer-Verlag.

OMG_1 (2003). *Common Warehouse Metamodel (CWM) Specification*. Version 1.1.

OMG_2 (2004). *Unified Modeling Language: Superstructure*. Version 2.0.

OMG_3 (2005). *Software Process Engineering Metamodel Specification*. Version 1.1.

OMG_4 (2005). *MOF 2.0/XMI Mapping Specification*. Version 2.1.

OMG_5 (2005). *MOF QVT Final Adopted Specification*.

OMG_6 (2006). *Meta Object Facility (MOF) Core Specification*. Version 2.0.

Patel-Schneider, P., P. Hayes, and I. Horrocks (2004). *OWL Web Ontology Language Semantics and Abstract Syntax*. Recommandation W3C.

Temis, *Insight Discoverer Categorizer*.
http://www.temis.com/fichiers/t_downloads/file_26_TEMIS_ID_categorizer_wp_v.2.0.pdf

Remerciements

Ces travaux ont été réalisés dans le cadre du projet RIAM « RelaxMultimedia » soutenu par les ministères de l'Industrie et de la Recherche. Ce travail a fait l'objet d'un partenariat avec l'Agence France Presse (AFP) et l'agence de presse RelaxNews.

Summary

In this paper, we show how techniques of Model Driven Engineering (MDE) can be used during the design process of ontology. Starting with an existing UML domain model, conveniently enriched with a set of stereotypes, we sketch the various steps of the ontology generation process. Using domain objects, this generation process also permits to populate the knowledge base, as structured by the ontology. Our work grounds on the ODM specification we briefly present here, in order to explain the main concepts that led our thought process. Our work is applied to the press domain and produces an ontology dedicated to navigating among the various pieces of information provided by newspaper dispatches and announcements.

DRAFT